Single-Dimensional Arrays

Introduction

A single array variable can reference a large collection of data.

Once an array is created, its size is fixed.

An array reference variable is used to access the elements in an array using an index.

Declaring Array Variables

To use an array in a program, you must declare a variable to reference the array and specify the array's *element type*. Here is the syntax for declaring an array variable:

```
elementType [ ]  arrayRefVar ;
```

The `elementType` can be any data type, and all elements in the array will have the same data type.

For example

```
double [ ] myList;
int [ ] x;
String [ ] s;
char [ ] ch;
```

Creating Arrays

The declaration of an array variable does not allocate any space in memory for the array. It creates only a storage location for the reference to an array.

If a variable does not contain a reference to an array, the value of the variable is `null`.

You cannot assign elements to an array unless it has already been created.

create an array by using the `new` operator and assign its reference to the variable with the following syntax:

```
arrayRefVar = new elementType [arraySize];
```

Declaring an array variable, creating an array, and assigning the reference of the array to the

variable can be combined in one statement as:

```
elementType [ ] arrayRefVar = new elementType [arraySize];
```

Here is an example of such a statement:

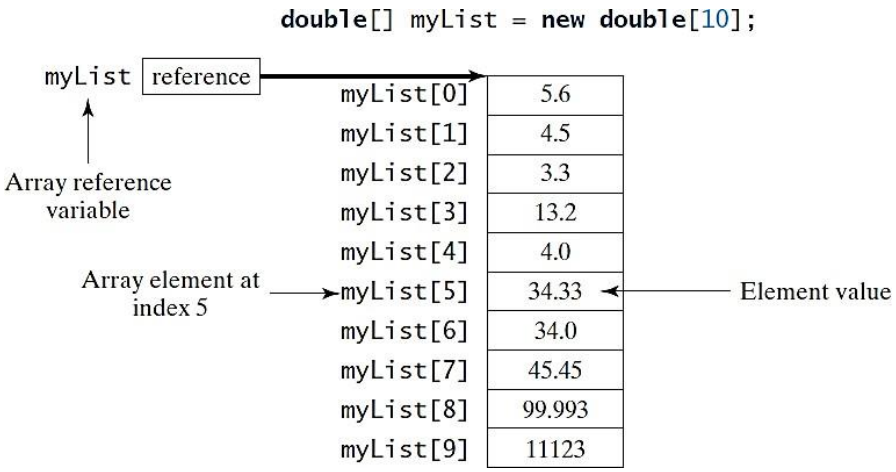
```
double [ ] myList = new double [10];
```

To assign values to the elements, use the syntax:

```
arrayRefVar [index] = value;
```

For example, the following code initializes the array.

```
myList[0] = 5. 6;
myList[1] = 4. 5;
myList[2] = 3. 3;
myList[3] = 13. 2;
myList[4] = 4. 0;
myList[5] = 34. 33;
myList[6] = 34. 0;
myList[7] = 45. 45;
myList[8] = 99. 993;
myList[9] = 11123;
```



Accessing Array Elements

The array elements are accessed through the index.

following syntax, known as an *indexed variable*:

```
arrayRefVar [index];
```

Array index start from 0 and the array index range from 0 to arrayRefVar.length-1.
For example,

myList[9] represents the last element in the array myList.

myList[3] represents the forth element in the array myList.

Array Initializers

The array initializer, which combines the declaration, creation, and initialization of an array in one statement using the following syntax:

elementType[] arrayRefVar = {value0, value1, ..., valuek};

For example, the statement

Double [ ] myList = {1.9, 2.9, 3.4, 3.5};

or

```
double[] myList = new double[4];
myList[0] = 1.9;
myList[1] = 2.9;
myList[2] = 3.4;
myList[3] = 3.5;
```

Note:

- The first element in an array with index 0.
- In a loop is using < should be used.
  For example, the following loop is wrong.

```
for (int i = 0; i < = list.length ; i++)
    System.out.print(list[i] + " ");
```

Processing Arrays

- you will often use a for loop—for two reasons:
- All of the elements in an array are of the same type.

Suppose you have the following array called mylist

```
double[ ] myList = new double [10];
```

The following are some examples of processing arrays.

1. Initializing arrays with input values:

```
java.util.Scanner input = new java.util.Scanner(System.in);
System.out.print("Enter " + myList.length + " values: ");
for (int i = 0; i < myList.length; i++) {
   myList[i] = input.nextDouble();
}
```

2. Initializing arrays with random values between 0.0 and 100.0:

```
for (int i = 0; i < myList.length; i++) {
    myList[i] = Math.random( ) * 100;
}
```

3. Displaying arrays (print an array elements):

```
for (int i = 0; i < myList.length ; i++) {
    System.out.print( myList [i] + " ");
}
```

4. Summing all elements of array:

```
double total =  0;
for (int i = 0; i < myList.length; i++) {
    total + =  myList[i];
}
```

5. Finding the largest element:

```
double max =  myList[0];
for (int i = 1; i < myList.length; i++) {
```

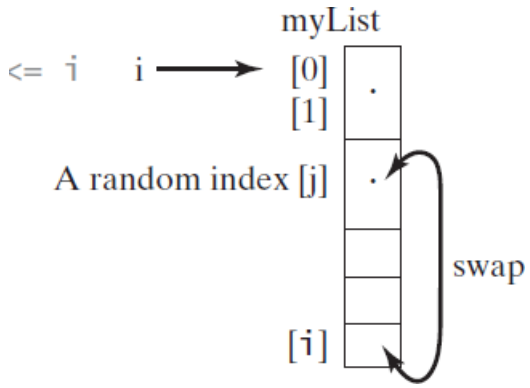
```
        if (myList[i] > max)
                max = myList[i];
}
```

6. Finding the smallest index of the largest element:
   Suppose the array myList is {1, 5, 3, 4, 5, 5}.

   The largest element is 5 and the smallest index for 5 is 1. Use a variable named max to store the largest element and a variable named indexOfMax to denote the index of the largest element.

```
double max = myList[0];
      int indexOfMax = 0;
      for (int i = 1; i < myList.length; i++) {
         if (myList[i] > max) {
            max = myList[i];
            indexOfMax = i;
         }
      }
```

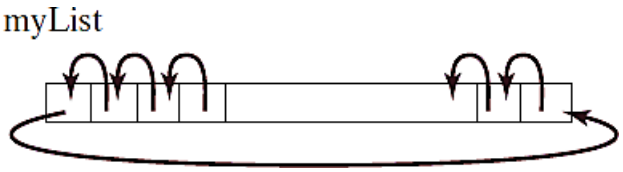
7. Random shuffling: In many applications, you need to randomly reorder the elements in an array. This is called shuffling. To accomplish this, for each element myList[i], randomly generate an index j and swap myList[i] with myList[j], as follows:




```
for (int i = myList.length  {}– 1; i > 0;i—) {
      // Generate an index j randomly with 0 <= j <= i
      int j = (int) (Math.random( ) * (i + 1) );
      // Swap myList[i] with myList[j]
         double temp = myList[i];
```

```
            myList[i] = myList[j];
            myList[j] = temp;
        }
```

Shifting elements:




```
double temp = myList[0];                    // store the first element in temp
for (int i = 1; i < myList.length; i++) {   // Shift elements left
    myList[i - 1] = myList[i];
}
myList[myList.length - 1] = temp;           //Move first element to last position
```

Write a program to read 100 numbers, get the average of these numbers, and find the number of the items greater than the average.

```
1  public class AnalyzeNumbers {
2    public static void main(String[] args) {
3      java.util.Scanner input = new java.util.Scanner(System.in);
4      System.out.print("Enter the number of items: ");
5      int n = input.nextInt();
6      double [] numbers = new double[n];
7      double sum = 0;
8
9      System.out.print("Enter the numbers: ");
10     for (int i = 0; i < n; i++) {
11       numbers[i] = input.nextDouble();
12       sum += numbers[i];
13     }
14
15     double average = sum / n;
16
17     int count = 0; // The number of elements above average
18     for (int i = 0; i < n; i++)
19       if (numbers[i] > average)
20         count++;
21
22     System.out.println("Average is " + average);
23     System.out.println("Number of elements above the average is "
24       + count);
25   }
26 }
```

numbers[0]
numbers[1]
numbers[2]
. create array
.
numbers[i]: .
numbers[n - 3]:
numbers[n - 2]: store number in array
numbers[n - 1]:
get average
above average?

```
Enter the number of items: 10 Enter
Enter the numbers: 3.4 5 6 1 6.5 7.8 3.5 8.5 6.3 9.5 Enter
Average is 5.75
Number of elements above the average is 6
```

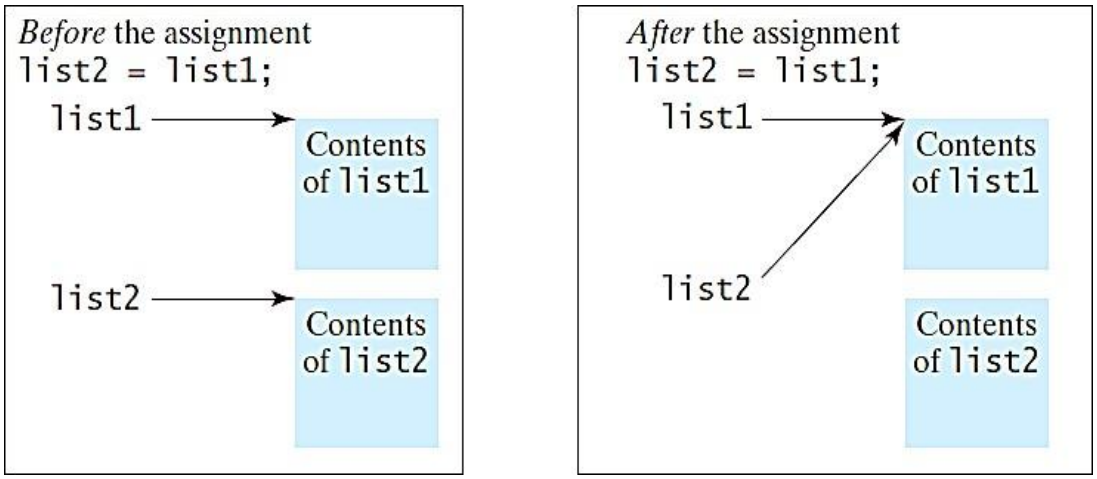
Copying Arrays

To copy the contents of one array into another, you have to copy the array's individual elements into the other array.

If you use the assignment statement (=), as follows:

```
list2 =  list1;
```

- this statement does not copy the contents of the array referenced by list1 to list2,
- this copies the reference value from list1 to list2. After this statement, list1 and list2 reference the same array, as shown in Figure bellow



In Java, you can use assignment statements to copy primitive data type variables, but not arrays. Assigning one array variable to another array variable actually copies one reference to another and makes both variables point to the same memory location.

There are two ways to copy arrays:

■ Use a loop to copy individual elements one by one.

```
int[ ] a1 = {2, 3, 1, 5, 10};
int[ ] a2 = new int [a1.length];
for (int i = 0; i < a1.length; i++) {
   a2[i] = a1[i];
}
```

■ Use the static arraycopy method in the *System* class. The syntax for arraycopy is:

arraycopy(sourceArray, srcPos, targetArray, tarPos, length);

For example:

System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length);

System.arraycopy(a1, 0, a2, 0, a1.length);

Passing Arrays to Methods

You can also pass arrays to methods.

For example:

```
public static void printArray( int[ ] array ) {
    for (int i = 0; i < array.length; i++) {
       System.out.print(array[i] + " ");
    }
  }
```

Example:

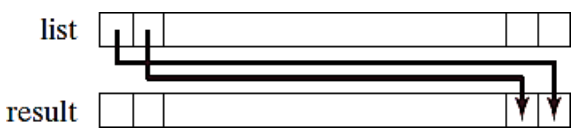
Define class contains two methods for swapping elements in an array. The first method, named swap, fails to swap two int arguments. The second method, named swapFirst-TwoInArray, successfully swaps the first two elements in the array argument.

```
1  public class TestPassArray {
2    /** Main method */
3    public static void main(String[] args) {
4      int[] a = {1, 2};
5
6      // Swap elements using the swap method
7      System.out.println("Before invoking swap");
8      System.out.println("array is {" + a[0] + ", " + a[1] + "}");
9      swap(a[0], a[1]);                                          false swap
10     System.out.println("After invoking swap");
11     System.out.println("array is {" + a[0] + ", " + a[1] + "}");
12
13     // Swap elements using the swapFirstTwoInArray method
14     System.out.println("Before invoking swapFirstTwoInArray");
15     System.out.println("array is {" + a[0] + ", " + a[1] + "}");
16     swapFirstTwoInArray(a);                                    swap array elements
17     System.out.println("After invoking swapFirstTwoInArray");
18     System.out.println("array is {" + a[0] + ", " + a[1] + "}");
19   }
20
21   /** Swap two variables */
22   public static void swap(int n1, int n2) {
23     int temp = n1;
24     n1 = n2;
25     n2 = temp;
26   }
27
28   /** Swap the first two elements in the array */
29   public static void swapFirstTwoInArray(int[] array) {
30     int temp = array[0];
31     array[0] = array[1];
32     array[1] = temp;
33   }
34 }
```

Returning an Array from a Method

A method may also return an array. For example,

```
public static int [ ] reverse ( int[ ] list ) {
    int [ ] result = new int[list.length];
    for ( int i = 0, j = result.length - 1;    i < list.length;    i++, j-- ) {
      result[j] = list[i];
    }
    return result;
  }
```



For each Loops

Java supports a convenient for loop, known as a foreach loop, which enables you to traverse the array sequentially without using an index variable. For example:

```
for ( double e : myList) {
   System.out.println( e );
}
```

In general, the syntax for a foreach loop is

```
for (elementDatatType   element : arrayRefVar ) {
            // Process the element
}
```

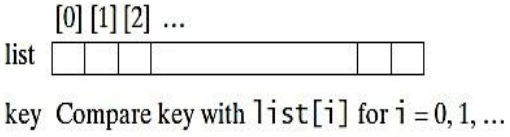
Searching Arrays

Two commonly used approaches, linear search and binary search.

- Searching is the process of looking for a specific element in an array—for example,
- If an array is sorted, binary search is more efficient than linear search for finding an element in the array.

The Linear Search Approach

```
1 public class LinearSearch {
2   /** The method for finding a key in the list */
3   public static int linearSearch(int[] list, int key) {
4     for (int i = 0; i < list.length; i++) {
5       if (key == list[i])
6         return i;
7     }
8     return -1;
9   }
10 }
```

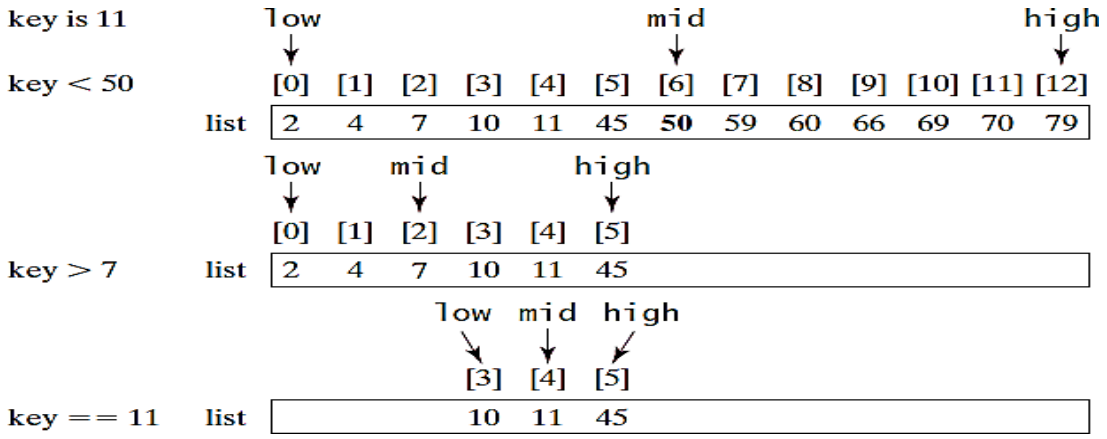


The Binary Search Approach

For binary search, the elements in the array must already be ordered (sorted).

The binary search first compares the key with the element in the middle of the array. Consider the following three cases:

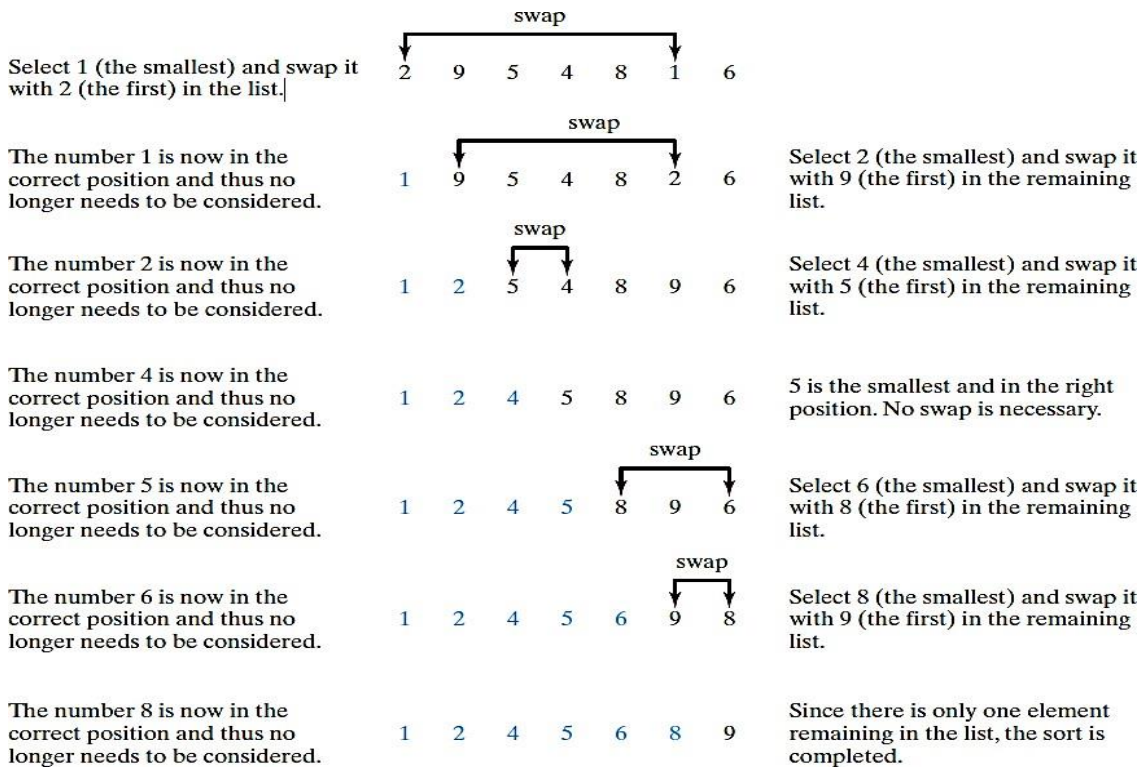
- If the key is less than the middle element, you need to continue to search for the key only in the first half of the array.
- If the key is equal to the middle element, the search ends with a match.
- If the key is greater than the middle element, you need to continue to search for the key only in the second half of the array.



```
1  public class BinarySearch {
2    /** Use binary search to find the key in the list */
3    public static int binarySearch(int[] list, int key) {
4      int low = 0;
5      int high = list.length - 1;
6
7      while (high >= low) {
8        int mid = (low + high) / 2;
9        if (key < list[mid])
10         high = mid - 1;
11       else if (key == list[mid])
12         return mid;
13       else
14         low = mid + 1;
15     }
16
17     return -low - 1; // Now high < low, key not found
18   }
19 }
```

Sorting Array (selection sort)

Figure bellow shows how to sort the list {2, 9, 5, 4, 8, 1, 6} using selection sort



```
import java.util.Scanner;

public class SelectionSort {
  static void print(int x[])
  {
     System.out.println("\n Display the elements of array one-D");
   for (int i = 0; i < x.length; i++) {
         System.out.print(x[i] + " ");
     }
  }
     static void read(int[] x) {
     Scanner input = new Scanner(System.in);
     for (int i = 0; i < x.length; i++) {
        x[i] = input.nextInt();
     }
   }
     static void selectionSort (int x[]){
        int temp;
        for(int i=0;i<x.length-1;i++)
            for(int j=i+1;j<x.length;j++)
               if(x[i]>x[j])
               {
                  temp=x[i];
                  x[i]=x[j];
                  x[j]=temp;
        }
     }

      public static void main(String[] args) {
     int[] list = new int[5];

     read(list);
     print(list);
     sort(list);
      print(list);
       }
}
```

بناء دالة لطباعة عناصر المصفوفة بشكل افقي

بناء دالة لادخال عناصر المصفوفة اثناء التنفيذ من console

بناء دالة ترتيب عناصر المصفوفة ترتيبا تصاعديا

الدالة الرئيسية

استدعاء دالة القراءة

استدعاء دالة الطباعة

استدعاء دالة الترتيب

استدعاء دالة الطباعة

Example 2:

```
import java.util.Scanner;

public class Arrayoperation {
    static void read(int[] x) {
    Scanner input = new Scanner(System.in);
    for (int i = 0; i < x.length; i++) {
       x[i] = input.nextInt();
    }
  }
    static void find(int x[],int y)
      {
       String a=" ";
       for(int i = 0; i < x.length; i++)
          if (y==x[i])
          {
             a="found";
       break;
          }
       else
             a="found";
      }
  public static void main(String[] args) {
    int[] list = new int[10];

    read(list);
    find(list,4);
    for (int i = 0; i < list.length; i++) {
       System.out.println("list["+i+"]=" +list[i]);
    }
  }
}
```

بناء دالة للبحث عن عنصر في المصفوفة اذا كان موجود يعرض found اذا غير موجود يعرض كلمة found